Towards a Constrained-based Verification of Parameterized Cryptographic Protocols

Najah Chridi, Mathieu Turuani, Michael Rusinowitch *

LORIA-INRIA (UMR 7503)
BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France
{chridi,turuani,rusi}@loria.fr

Abstract. Although many works have been dedicated to standard protocols like Needham-Schroeder very few address the more challenging class of group protocols. We present a synchronous model for group protocols, that generalizes standard protocol models by permitting unbounded lists inside messages. In this extended model we propose a correct and complete set of inference rules for checking security properties in presence of an active intruder for the class of well-tagged protocols. Our inference system generalizes the ones that are implemented in several tools for a bounded number of sessions and fixed size lists in message. In particular when applied to protocols whose specification does not contain unbounded lists our inference system provides a decision procedure for secrecy in the case of a fixed number of sessions.

1 Introduction

Cryptographic protocols are crucial for securing electronic transactions. They rely on cryptographic functions to ensure security properties such as secrecy or authentication. The confidence in these protocols can be increased by a formal analysis in order to verify that the security properties are met at least at the logical level, that is, even when abstracting from the cryptographic functions and considering messages as first-order terms. Verification at the logical level is nevertheless a non-trivial task since cryptographic protocols are infinite state systems and for instance the set of potential messages that can be generated by an intruder is unbounded. Recently numerous works have been dedicated to the design of automated verification tools for cryptographic protocols. Such tools are often based on model-checking, modal logics, equational reasoning, and resolution theorem-proving (see e.g., [Wei99,AC02,MT07]). Checking whether a protocol is flawed in the abstract Dolev Yao model [DY83] can often be reduced to a constraint solving problem in a term algebra (modulo an equational theory). This constraint-based approach has proved to be quite effective on standard benchmarks and has also permitted the discovery of new flaws in several protocols (see e.g., [BMV03]).

* This work was supported by the FP7-ICT-2007-1 Project no. 216471, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures"(www.avantssar.eu) and by CPER Project SeCoManet.

However to our knowledge it has never been applied to the more challenging group protocols. In fact very few formal verification results are available for such protocols. The difficulty relates to the fact that group protocols may perform an arbitrary number of steps since the group of communicating agents is a priori unbounded. This allows one to encode easily undecidable problems.

Related work Several works have considered protocols with unbounded number of participants and recursive steps. The formal analysis of such protocols goes up with Paulson in [Pau97] who studied the Recursive Authentication (RA) protocol [BO97] for an unbounded number of participants using Isabelle/HOL theorem prover [Pau96]. However if the protocol is defective there is no automatic mechanism to find the attack.

The validation of group protocols has been investigated in the CLIQUES project [SWT98], based on group Diffie-Hellman (A-GDH) protocols. Several analysis methods have been applied in this project, from manual to automatic ones. An interesting result has been obtained by Pereira and Quisquater [PQ03] who found several attacks on the CLIQUES suite and have shown that it is impossible to design a correct authentication group key agreement protocol built on A-GDH for a number of participants greater than three [PQ04]. Recently Kremer and al. [KMT08] have developped an automata-based approximation technique to analyse this class of protocols and check the absence of flaw in presence of a **passive intruder**. In [MS01], the NRL protocol analyser, which is based on a combination of model checking and theorem-proving techniques, has been adapted to handle the GDOI's protocols. Although Diffie-Hellman exponentiation has been encoded in the tool, it was not able to rediscover Pereira-Quisquater attacks on the CLIQUES suite [Mea00]. Coral system [SB04] has analysed an improved version of the multicast group key management protocol by Tanaka and Sato [TJ03]. Two serious attacks have been found on this protocol. Coral has also discovered other attacks on Asokan-Ginzboorg [AG00] and Iolus [Mit97] protocols.

Some works have focused on the modelling of recursive computations performed by some participants (such as a server) in group protocols. In [KW04] tree transducers are introduced to model recursion and to allow the protocol participants to output structured messages. This work gives a decision algorithm for secrecy in the case of atomic keys and bounded message size in the Dolev Yao setting. However messages cannot be tested for equality without losing decidability. Similarly using composed keys or adding equational theories for XOR or Diffie-Hellman exponentiation in their model leads to undecidability. In [Tru05], Truderung introduces a class of Horn clauses to model the recursive behavior of participants. In this model protocol participants may receive messages of unbounded sizes, send multiple messages in a single step, compare and store messages. He gives a decision procedure to check whether protocols in this model satisfy secrecy properties. His algorithm is in NEXPTIME and is based on the derivation of an exponential bound on the size of minimal attacks. Hence this nice result is rather of theoretical flavour and is not suitable for an implementation. Only atomic keys are allowed for encryption. Moreover, Truderung's model cannot model some computations such as list mapping or

functional symbol mapping. Note that non-atomic keys can be handled by our verification procedure (to be presented in the following sections). In [KT07] an extension of [Tru05] has been proposed to handle XOR operator. Security can then be decided for a class of recursive protocols where principals are forbidden to XOR several messages (depending on messages) received from the network. Another extension [KKW07] has been designed to model freshness of nonces and keys more accurately.

Contribution. We present a synchronous model for parameterized protocols, that can be viewed also as an extension of classical protocols model to handle uniform lists of messages whose length is a fixed parameter. Such kind of protocols can be found in different domains. As example, we can cite web services protocols where lists of messages are commonly used. We can consider for instance, a basic action of an honest participant who received a messages list whose length is not known in advance and sends a message based on the informations he got:

$$\langle \{m_1\}_k, .., \{m_i\}_k, .., \{m_n\}_k \rangle \longrightarrow f(\langle m_1, .., m_i, .., m_n \rangle)$$

In this step, the participant decrypts the different messages composing the received list using the public key of the sender. He then builds the message to be sent using the list of messages m_i and a function f. Note that this kind of protocols are used in web services where messages may contain unbounded lists of encrypted XML nodes. Moreover, our model can handle in particular, group protocols that have an unbounded number of participants, and thus, admit this number as a parameter. For this model we propose a complete and correct set of inference rules that allows one to check the security of *well-tagged* protocols in this class in presence of an **active** intruder. We show that relaxing the conditions on well-tagged protocols leads immediately to undecidability. Moreover, the proposed rules can model **list mapping** and allow **non atomic keys**. The class of protocols that we study admits tagged messages. Tagging basically avoids some unifications between messages that could be exploited for attacks. Several works on protocols have considered tagging techniques on messages as ours in order to enforce decidability. But these works ([BP03,RS03]) do not consider group protocols, or protocol with unbounded lists. Moreover in our case tagging is limited to messages that contains indexed variables, that is variables to be instanciated by items of unbounded lists. The other messages do not need to be tagged.

Organization of the paper. We introduce in Section 2 our protocol model. We define attacks and show that their detection is undecidable. This motivate us to introduce the class of well-tagged protocols that is a good candidate for decidable security. In Section 3, we introduce auxiliary predicates and their semantics. They are meant to express message constructibility (from intruder knowledge) and they are used to build constraint system whose satisfiability is equivalent to the existence of attacks on a protocol. In Section 4, we introduce a set of simplification rules to reduce these constraints. Finally, in Section 5, we give an algorithm for applying these rules on a constraint system modelling a protocol security problem. Full details with proofs of correctness, completeness and decidability of normalized constraints are given in a technical report [Rep].

1.1 Motivating Example: Synchronous Group Protocols

As a motivating and running example, we introduce the Asokan-Ginzboorg group protocol which is an application level protocol. Let the group be of size $n+1$ for $n \geq 1$. The protocol describes the establishment of a session key between a leader (a_{n+1}) and a random number n of participants ($a_i, i = 1..n$). Indeed, the leader starts the communication by sending a key (e). As a response, each participant generates a symmetric key (r_i) and a contribution to the group key (s_i) and sends them encrypted under e. The group key would be $f(s_1, \ldots, s_{n+1})$.

1. $a_{n+1} \rightarrow \text{ALL} : \langle a_{n+1}, \{e\}_p \rangle$
2. $a_i \rightarrow a_{n+1} : \langle a_i, \{\langle r_i, s_i \rangle\}_e \rangle \quad 1 \leq i \leq n$
3. $a_{n+1} \rightarrow a_i : \{\langle s_1, .., s_{n+1} \rangle\}_{r_i} \quad 1 \leq i \leq n$
4. $a_i \rightarrow a_{n+1} : \langle a_i, \{\langle s_i, h(s_1, \ldots, s_{n+1}) \rangle\}_K \rangle$ some i, $K = f(s_1, \ldots, s_{n+1})$

Since here, modulo index renaming, all members of the group have identical actions, we are going to abstract them in one agent as it shows Figure 1.1.
This Figure shows the transformation between an initial version of a protocol

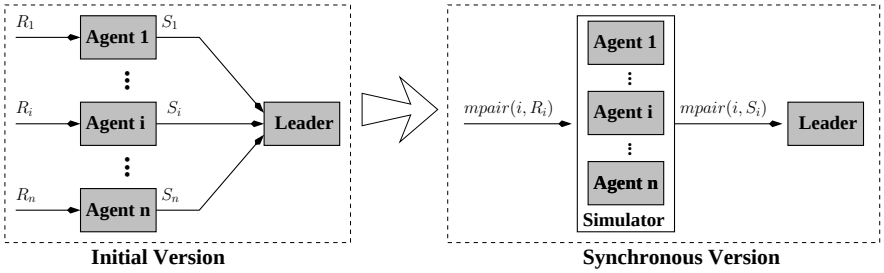

Fig. 1. Transformation to a Synchronous Protocol

towards a synchronous version. In the initial version, agents receive from the leader some message R_i (for the agent i) and send the message S_i. Messages R_i have the same pattern, as well as messages S_i. We abstract all these agents into a single agent : the simulator who, while receiving the list of R_i represented by the variadic list constructor $mpair(\_, \_)$., sends another list containing S_i represented by the same operator. This protocol is called *synchronous protocol*. For our example, the simulator is called S and it simulates agents $a_1, \ldots, a_n$. Agent L simulates the leader a_{n+1}. This way, we obtain below a protocol with a fixed number of steps to the expense of introducing a variadic list constructor $mpair(\_, \_)$:

1. $L \longrightarrow S : mpair(t, \langle l, \{e\}_p \rangle)$
2. $S \longrightarrow L : mpair(i, \langle a_i, \{\langle r_i, s_i \rangle\}_e \rangle)$
3. $L \longrightarrow S : mpair(i, \{\langle mpair(j, s_j), s' \rangle\}_{r_i})$
4. $S \longrightarrow L : mpair(i, \langle a_i, \{\langle s_i, h(\langle mpair(k, s_k), s' \rangle) \rangle\}_{f(\langle mpair(k, s_k), s' \rangle)} \rangle)$

Note that All parametric lists have the same length n.

2 The Protocol Model

We extend the protocol model [RT03] in order to deal with parametric lists (whose length is the parameter). They are constructed with a new operator denoted by $mpair(\_,\_)$. The intuition is that a $mpair$ message is equivalent to a list of messages built with the same pattern.

2.1 Names, Operators and Messages

Let $\mathcal{X}$ be a set of variables represented by capital letters. Let $\mathcal{I}$ be a countable set of index variables. Let $\overrightarrow{\mathcal{X}}$ be a set of symbols represented by overarrowed capital letters, disjoint from $\mathcal{X}$. Let $\mathcal{X}_\mathcal{I} = \{Y_i \text{ s.t. } \overrightarrow{Y} \in \overrightarrow{\mathcal{X}} \text{ and } i \in \mathcal{I}\}$ be a countable set of (indexed) variables. Similarly, let $\mathcal{C}$ and $\overrightarrow{\mathcal{C}}$ be (disjoint) sets of symbols, represented by (overarrowed) lower case letters, and let $\mathcal{C}_\mathcal{I} = \{c_i \text{ s.t. } c \in \overrightarrow{\mathcal{C}} \text{ and } i \in \mathcal{I}\}$. Elements in $\mathcal{C}$ and $\mathcal{C}_\mathcal{I}$ are called constants. In this paper, terms can be (optionally) tagged by an index. That is, let $\overrightarrow{e} \in \overrightarrow{\mathcal{C}}$ be a symbol that we reserve for tagging operations only. Then a term is an element of $\mathcal{T}$ in the following language:

$$\begin{array}{l} \mathcal{T}_s = \{\mathcal{T}\}^p_\mathcal{T} \mid \{\mathcal{T}\}^s_\mathcal{T} \mid h(\mathcal{T}) \mid \langle \mathcal{T}, \mathcal{T} \rangle \mid mpair(\mathcal{I}, \mathcal{T}) \mid \mathcal{X} \mid \mathcal{X}_\mathcal{I} \mid \mathcal{C} \mid \mathcal{C}_\mathcal{I} \backslash \{e_i \mid i \in \mathcal{I}\} \\ \mathcal{T} = [e_i, \mathcal{T}_s] \mid \mathcal{T}_s \qquad \text{with } i \in \mathcal{I} \end{array}$$

where $\mathcal{T}_s$ is by definition the set of untagged terms. The operators $\{\_\}^p_\_$ and $\{\_\}^s_\_$ represent asymmetric and symmetric encryptions respectively, $\langle \_, \_ \rangle$ is a pairing operator, h is a hash function and $mpair(i,t)$ is a symbolic representation of a list (or tuple) of terms, built from the common pattern t by iterating i along integers. The translation function defined in Section 2.2 gives the semantics of this operator. We denote by signature $\mathcal{G}$ the set of operators in $\mathcal{T}_s$. To simplify the syntax, in the following we will write t^i instead of $[e_i, t]$, and call it a *tagged term*. We also omit the tag i of a term t^i, whenever the tag i is not relevant to the discussion. We denote by $\mathcal{T}_g$ the set of ground terms, i.e. any term $t \in \mathcal{T}$ with no variable in $\mathcal{X}$ or $\mathcal{X}_\mathcal{I}$ and no $mpair$ symbol. Ground terms will be used to describe messages that are circulated in a protocol run. Given a term t we denote by $Var(t)$ (resp. $Cons(t)$) the set of variables (resp. constants) occuring in t. We denote by $Atoms(t)$ the set $Var(t) \cup Cons(t)$.

In order to represent a list of terms we iterate the pairing operator $\langle \_, \_ \rangle$. For instance to represent a, b, c, d we can use the term $\langle a, \langle b, \langle c, d \rangle \rangle \rangle$ and we shall write this term in a shorthand: $\langle a, b, c, d \rangle$. However we do not assume any associativity property of pairing.

A substitution σ assigns terms to variables. A ground substitution assigns ground terms to variables. The application of σ to a term t is written $t\sigma$. These notations are extended to sets of terms E in a standard way: $E\sigma = \{t\sigma | t \in E\}$. The set of subterms of t is denoted by $Subterm(t)$. It is defined recursively as follows: If t is a variable or a constant then $Subterm(t) = \{t\}$. If $t = f(t_1, .., t_n)$ or $t = f(t_1, .., t_n)^i$ with $t \in \mathcal{G}$, then $Subterm(t) = \{t\} \cup \cup_{i=1}^n Subterm(t_i)$. Note that

u is *not* considered as a subterm of u^i. We denote by $\leq$ the subterm relation on $\mathcal{T}$. We define the relation $\leqslant_m$ over $\mathcal{T} \times \mathcal{T}$ as the smallest reflexive and transitive relation such that if $t = f(t_1, .., t_n)$ or $t = f(t_1, .., t_n)^j$ with $f \neq mpair$, then for all $i = 1, \ldots, m$ we have $t_i \leqslant_m f(t_1, .., t_m)$. Note that $t \leqslant_m u$ implies $t \leq u$.

Finally, we define two kind of Index-operations: replacements, used in the inference rules over constraints, and substitutions, used to define the solutions of constraints (See Section 3).

Definition 1 (Index-Replacement δ and Index-Substitution τ). *An Index-Replacement δ (resp. Index-Substitution τ) is an application from $\mathcal{I}$ to $\mathcal{I}$ (resp. to non-negative integers) that is extended to indexed variables and constants with $\delta(X_i) = X_{\delta(i)}$ and $\delta(c_i) = c_{\delta(i)}$ (resp. $\tau(X_i) = X_{\tau(i)}$ and $\tau(c_i) = c_{\tau(i)}$) and extended to terms and sets of terms in the natural way.*

We will use the notations $\delta_{i,j}$ (resp. $\tau_{i,j}$) to denote the replacement (resp. substitution) of $i \in \mathcal{I}$ by $j \in \mathcal{I}$ (resp. $j \in \mathbb{N}$). We also use $\delta^k_{i,j}$ to denote the replacement of $i \in \mathcal{I}$ by $j \in \mathcal{I}$ and the other indexes apart from i by $k \in \mathcal{I}$. We define the set of indexes occuring in a term as follows:

Definition 2 (Term Indexes). *Given a term $t \in \mathcal{T}$, we denote by $Var_{\mathcal{I}}(t)$ the set of indexes in t, recursively defined as follows:*

$Var_{\mathcal{I}}(mpair(i,t)) = Var_{\mathcal{I}}(X) = Var_{\mathcal{I}}(c) = \emptyset$ *with* $X \in \mathcal{X}$ *and* $c \in \mathcal{C}$
$Var_{\mathcal{I}}(X_i) = Var_{\mathcal{I}}(c_i) = \{i\}$ *with* $X_i \in \mathcal{X}_{\mathcal{I}}$ *and* $c_i \in \mathcal{C}_{\mathcal{I}}$
$Var_{\mathcal{I}}(f(t_1, .., t_n)) = Var_{\mathcal{I}}(t_1) \cup .. \cup Var_{\mathcal{I}}(t_n)$ *otherwise*
$Var_{\mathcal{I}}(t^i) = Var_{\mathcal{I}}(t) \cup \{i\}$

2.2 Protocol Specification, Intruder and attacks

A protocol is given by a set of principals and a finite list of steps for each. We associate to each principal A a partially ordered finite set $(W_A, <_{W_A})$ steps $R_\iota \Rightarrow S_\iota$ where R_ι is an expected message and S_ι his reply. $Init$ and End are fixed messages used to initiate and close a protocol session. Our notion of correct execution of a protocol session (or *protocol run*) follows [RT03].

We follow the intruder model of Dolev and Yao [DY83]. The actions of the intruder are simulated by a sequence of rewrite rules on sets of messages. These rules are defined as follows. We note $\longrightarrow^*_{DY}$ their reflexive and transitive closure.

Decompositition Rules
$L_d(\langle a_1, \ldots, a_n \rangle) : \langle a_1, \ldots, a_n \rangle \rightarrow a_1, \ldots, a_n, \langle a_1, \ldots, a_n \rangle$
$L_d(\{a\}^p_K) : \{a\}^p_K, K^{-1} \rightarrow \{a\}^p_K, K^{-1}, a$
$L_d(\{a\}^s_K) : \{a\}^s_K, b \rightarrow \{a\}^s_K, b, a$
$L_d(t^i) : t^i \rightarrow t$

Compositition Rules
$L_c(\langle a_1, \ldots, a_n \rangle) : a_1, \ldots, a_n \rightarrow a_1, \ldots, a_n, \langle a_1, \ldots, a_n \rangle$
$L_c(\{a\}^p_K) : a, K \rightarrow a, K, \{a\}^p_K$
$L_c(\{a\}^s_K) : a, b \rightarrow a, b, \{a\}^s_K$
$L_c(t^i) : t \rightarrow t^i$ for any $i \in \mathcal{I}$

Definition 3 (Non-Redundant Derivation). *Let $D = E_0 \longrightarrow_{L_1} \ldots \longrightarrow_{L_l} E_l$ be a derivation such that $u \in E_l$ and $\forall i < l \; u \notin E_i$. We call u a goal of D and we note D as $D_u(E_0)$. D is a non redundant derivation if $\forall i \, \forall t \in E_i$, if $L_c(t) \in D$ then $\nexists L_d(\_) \in D$ that generates t, and if $\exists L_d(\_) \in D$ generating t then $L_c(t) \notin D$. We denote by NRD the set of non-redundant derivations.*

Remark 1. For each derivation $D_t(E)$, there exists a non redundant derivation $D'_t(E)$. Indeed, D' is obtained by elimination of $L_c(t)$ if $L_d(\_) \in D$ where $L_d(\_)$ generates t and by eliminating each $L_d(\_) \in D$ that generates t if $L_c(t) \in D$.

We define a predicate Dy. This predicate checks whether a message can be constructed by the intruder from some known messages.

Definition 4 (Dy, Dy_c and Dy_d). *Let E and $\mathcal{K}$ be sets of ground terms and t be a ground term such that there exists $D \in NRD$ with goal t without decomposing any term of $\mathcal{K}$. Then, we say that t is forged from E and we denote it by $t \in Dy(E, \mathcal{K})$. Moreover, if $D = D'.L_c(t)$ then $t \in Dy_c(E, \mathcal{K})$, otherwise $t \in Dy_d(E, \mathcal{K})$.*

We interpret the $mpair(\_,\_)$ operator in the standard Dolev-Yao signature by defining a translation function that replaces any *mpair* by a sequence of *pair* applications. The number of such applications is given as a parameter e to the translation function. The integer represents the common length of lists of terms represented by any $mpair(,)$.

Definition 5 (Translation of terms). *Let $\mathcal{T}_{DY}$ be the set of terms without any $mpair(,)$. Given any integer e and function $f \neq mpair$, the function $^{-_e}$ from $\mathcal{T}$ to $\mathcal{T}_{DY}$ is defined as follows:*

$$\overline{mpair(i,t)}^e = \langle \overline{\tau_{i,1}(t)}^e, .., \overline{\tau_{i,e}(t)}^e \rangle \text{ and } \overline{f(s_1,..,s_k)}^e = f(\overline{s_1}^e, .., \overline{s_k}^e)$$

We can now define attacks on protocols in our model, based on Dy predicate.

Definition 6 (Attack). *Given a protocol execution $P = \{R_i \Rightarrow S_i | i = 1, .., k\}$, a secret Sec and assuming the intruder has as initial knowledge S_0, an attack is described by a ground substitution σ, an Index-Substitution τ, and an integer e, such that $\forall i = 1, \ldots, k$, we have:*

$$\overline{R_i\tau\sigma}^e \in Dy(\overline{S_0}^e, \overline{S_1}^e\tau\sigma, \ldots, \overline{S_{i-1}}^e\tau\sigma, \emptyset)$$
$$\overline{Sec}^e \in Dy(\overline{S_0}^e, \overline{S_1}^e\tau\sigma, \ldots, \overline{S_k}^e\tau\sigma, \emptyset)$$

2.3 Undecidability and Well-Tagged Protocols

Unfortunately, the insecurity problem (i.e. the existence of an attack) is undecidable in the general case. This can be shown by encoding Post Correspondance Problem (PCP) with two letters. Note that this encoding requires only atomic keys.

Definition 7 (PCP protocol). *Let $J = \{(\alpha_1, \beta_2), .., (\alpha_p, \beta_p)\}$ be an instance of PCP on the alphabet $\{a, b\}$. We define the protocol specification $P(J)$ coding J as the following, with $\mathcal{C} = \{a, b, 0, t, u\}$, $\mathcal{X} = \{Z\}$, $\overrightarrow{\mathcal{X}} = \left\{\overrightarrow{A}, \overrightarrow{B}, \overrightarrow{X}, \overrightarrow{Y}\right\}$ and only one honest participant :*

1. $Init \Rightarrow a,\ b,\ 0,\ \{\langle 0, 0\rangle\}_t$
2. $mpair(i, \langle A_i, B_i\rangle) \Rightarrow mpair(i, \{\langle A_i, B_i\rangle\}_t)$
3. $mpair(i, \{\langle X_i, Y_i\rangle\}_t) \Rightarrow mpair(i, \{\langle \alpha_1(X_i), \beta_1(Y_i)\rangle\}_u), \ldots$ $mpair(i, \{\langle \alpha_p(X_i), \beta_p(Y_i)\rangle\}_u)$
4. $mpair(i, \{\langle A_i, B_i\rangle\}_u), \{\langle Z, Z\rangle\}_u \Rightarrow Sec$

Theorem 1. *An instance J of PCP has a solution iff $P(J)$ has an attack on Sec.*

The proof of this theorem is detailed in technical report [Rep]. We will therefore introduce the class of Well-Tagged protocols for which decidability is expected. To do this, we first introduce the notion of autonomy:

Definition 8 (Autonomy). *A term $mpair(i, u)$ is sayed to be autonomous when $Var_{\mathcal{I}}(u) \subseteq \{i\}$. A term $t \in T_{DY}$ is autonomous if $\#Var_{\mathcal{I}}(t) \leq 1$ and $\forall t' < t$, t' is autonomous. A protocol $\mathcal{P} = \{R_\imath \Rightarrow S_\imath | \imath \in J\}$ is autonomous iff for all $\imath \in J$, $R_\imath$ and $S_\imath$ are autonomous and $Var_{\mathcal{I}}(R_\imath) = \emptyset$ and $Var_{\mathcal{I}}(S_\imath) = \emptyset$.*

For instance, the term $t = mpair(i, mpair(j, \{a_i\}_{c_j}))$ is not autonomous. We remark that the autonomy property alone is not enough to guarantee decidability, since the PCP protocol of Definition 7 is autonomous.

Definition 9 (Well-Tagged protocols). *$\mathcal{P} = \{R_\imath \Rightarrow S_\imath | \imath \in J\}$ is Well-Tagged iff:*

1. *$\forall \imath \in J$, $\forall X_i \in \mathcal{X}_{\mathcal{I}} \cap Subterm(R_\imath) \cap \bigcup_{\imath' < \imath} Subterm(R_{\imath'})$, X_i is tagged;*
2. *$\forall \imath \in J$, $\forall X_i \in \mathcal{X}_{\mathcal{I}} \cap Subterm(S_\imath)$, X_i is tagged;*
3. *$\forall \imath \in J$, $\forall t = f(s_1, .., s_k) \in Subterm(S_\imath)$ with $f \neq mpair$, if $\exists j = 1..k$ s.t. s_j is tagged, then t is tagged too;*
4. *$\mathcal{P}$ is autonomous.*

In this definition, Conditions 1 and 2 state that any indexed variable of the protocol must be tagged, except for its first occurrence w.r.t. the partial step ordering. Moreover, Condition 3 (when combined with Condition 2) states that, for any subterm t of some acquired intruder knowledge $S_\imath$, if an indexed variable is accessible from t by decompositions without opening any $mpair$, then t must be tagged. Note that as a consequence of mpair autonomy, an indexed variable X_i can only be tagged by its index (as in $X_i{}^i$) or untagged.

The idea underlying the tagging of variables is to add enough information on terms in $mpair$ so that the protocol cannot be used to test or ensure relations between elements of the same $mpair$, such as $\forall i = 2..n$, $\exists i' = 1..n$ s.t. $X_i = f(X_{i'})$. This is precisely the kind of relations that the encoding of PCP is able to exploit. Thus, adding tags to the PCP-encoding protocol will generate a new protocol that cannot be run.

3 Constraints for Protocol Verification

We will use a symbolic constraint system to represent all runs of a protocol given a step ordering. This system uses (universal or existential) quantifiers on index variables and includes an (implicit) universal quantification on the number n of elements in any $mpair$. Before defining our constraint system, some basic notions have to be introduced. For terms s, s' (resp. sets of terms E, E') we note $s \sim s'$ (resp. $E \sim E'$) if they are equal once we erase their tags.

Definition 10 (Relation $\leqslant_E^L$ for accessible subterms). *We consider a relation $\_\leqslant\_$ on $\mathcal{T} \times 2^{\mathcal{T}} \times 2^{\mathcal{T}} \times \mathcal{T}$. We write $s \leqslant_E^L t$ for s, t terms in $\mathcal{T}$ and E and L finite subsets of $\mathcal{T}$. Note that this can be used for $\mathcal{T}_s$ too. This relation is defined as the smallest relation such that:*

$$
\begin{array}{ll}
t \leqslant_\emptyset^\emptyset t & \forall t \in \mathcal{T} \\
s \leqslant_E^L t & \textit{iff } s' \leqslant_{E'}^{L'} t' \textit{ where } s \sim s', t \sim t', E \sim E', L \sim L' \\
\textit{If } \{m\}_k^p \leqslant_E^L t & \textit{then } m \leqslant_{E,k^{-1}}^{L'} t \textit{ where } L' = L \cup \{\{m\}_k^p\} \\
\textit{If } \{m\}_b^s \leqslant_E^L t & \textit{then } m \leqslant_{E,b}^{L'} t \textit{ where } L' = L \cup \{\{m\}_b^s\} \\
\textit{If } \langle t_1, \ldots, t_n \rangle \leqslant_E^L t & \textit{then } \forall i \leq n, t_i \leqslant_E^{L'} t \textit{ where } L' = L \cup \{\langle t_1, \ldots, t_n \rangle\} \\
\textit{If } m \leqslant_{E'}^L t \textit{ and } E' \subset E & \textit{then } m \leqslant_E^L t
\end{array}
$$

We write $u \leqslant_E t$ when $u \leqslant_E^L t$ for some L. Note that by construction, $u \leqslant_E t$ implies $u \in Subterm(t)$. We say that u is a subterm of t that is accessible, i.e. can be obtained from t by decompositions using keys in E. For simplicity, we note $\leqslant_{b_1,..,b_k}$ instead of $\leqslant_{\{b_1,..,b_k\}}$. We define also the set of strict accessible terms by $s <_F t$ (resp. $s <_F^L t$) if $s \leqslant_F t$ (resp. $s \leqslant_F^L t$) and $s \neq t$. Given $t \leqslant_F^L u$ or $t <_F^L u$), we call length of $t \leqslant_F^L u$ or (resp $t <_F^L u$) the number of elements of L.

Before defining constraint systems, we need to introduce the environment and elementary constraints.

Definition 11 (Environment). *We call an* environment *a finite set of equalities $X = u$ whose left-hand sides are variables ($X \in \mathcal{X} \cup \mathcal{X}_\mathcal{I}$). We usually denote it by $\mathcal{E}$.*

Definition 12 (Elementary Constraint). *An elementary constraint is an expression $(t = t')$, $(t \in Forge(E, \mathcal{K}))$, $(t \in Forge_c(E, \mathcal{K}))$, $(t \in Sub(t', E, \mathcal{E}, \mathcal{K}))$, or $(t \in Sub_d(t', E, \mathcal{E}, \mathcal{K}))$ with an environment $\mathcal{E}$, $t, t' \in \mathcal{T}$ and $E \subset \mathcal{T}$.*

An elementary constraint represents a basic relation on terms. Indeed, $t \in Forge(E, \mathcal{K})$ if the term t is derivable from the knowledge E without decomposing elements of $\mathcal{K}$; $t \in Forge_c(E, \mathcal{K})$ if t is derived by composition; $t \in Sub(t', E, \mathcal{E}, \mathcal{K})$ if t is an accessible subterm from t' with knowledge E with none of the intermediate terms between t and t' in $\mathcal{K}$, and this modulo replacements using equations of $\mathcal{E}$; $t \in Sub_d(t', E, \mathcal{E}, \mathcal{K})$ if t is accessible by decomposition of t', also modulo replacements using $\mathcal{E}$; and $t = t'$ if t and t' are equal.

Definition 13 (Negative Constraint). *A negative constraint is an expression $(\forall i\ X_m \neq u)$ or $(X_m \notin Forge_c(E, \mathcal{K}))$ with $X_m \in \mathcal{X}_\mathcal{I}$, $u \in \mathcal{T}$, $E, E' \subset \mathcal{T}$ and $i \in Var_\mathcal{I}(u)$.*

The set of solutions of a constraint S, denoted by $[\![S]\!]_\tau^e$ where e is a value of n and τ is an Index-Substitution is a set of ground substitutions to be defined in the following. We define $\mathcal{G}S$ to be the set of all ground substitutions.

Definition 14 (Solutions of an Elementary Constraint).

$$
\begin{array}{ll}
[\![t = t']\!]_\tau^e & = \{\sigma \in \mathcal{G}S \mid \overline{t}^e \tau\sigma = \overline{t'}^e \tau\sigma\} \\
[\![t \in Forge(E, \mathcal{K})]\!]_\tau^e & = \{\sigma \in \mathcal{G}S \mid \overline{t}^e \tau\sigma \in Dy(\overline{E}^e \tau\sigma, \overline{\mathcal{K}}^e \tau\sigma)\} \\
[\![t \in Forge_c(E, \mathcal{K})]\!]_\tau^e & = \{\sigma \in \mathcal{G}S \mid \overline{t}^e \tau\sigma \in Dy_c(\overline{E}^e \tau\sigma, \overline{\mathcal{K}}^e \tau\sigma)\}
\end{array}
$$

$$
\begin{array}{l}
[\![\, t \in Sub(w, E, \mathcal{E}, \mathcal{K}) \,]\!]^e_\tau = \\
\qquad \{\sigma \in \mathcal{GS} \mid \exists u\ \exists F, L \ s.t.\ u \leqslant^L_F \overline{w}^e\tau,\ F\sigma \subseteq Dy(\overline{E}^e\tau\sigma, \overline{\mathcal{K}}^e\tau\sigma), \\
\qquad L\sigma \cap \overline{\mathcal{K}}^e\tau\sigma = \emptyset,\ \text{and either } u\sigma = \overline{t}^e\tau\sigma \text{ or } \exists v, \delta, k,\ \tau' \ s.t \\
\qquad \left\{ \begin{array}{l}
\left\{ \begin{array}{l}
\text{either } u \in \mathcal{X},\ (u = v) \in \mathcal{E},\ \delta = \emptyset,\ \text{and } \tau' = \tau \\
\text{or } \exists \overrightarrow{Z} \in \overrightarrow{\mathcal{X}},\ i, j \in \mathcal{I} \ s.t\ u = Z_i\tau',\ (Z_j = v) \in \mathcal{E}, \\
\quad \delta = \delta^k_{j,i}, \tau \subseteq \tau' \text{ and } \mathcal{D}om(\tau') = \mathcal{D}om(\tau) \cup \{k, i\}
\end{array} \right. \\
k, i \notin Var_{\mathcal{I}}(\{t, w, \mathcal{E}\}), u\sigma \notin Dy_c(\overline{E}^e\tau\sigma, \overline{\mathcal{K}}^e\tau\sigma) \\
u\sigma = \overline{v}^e\delta\tau'\sigma,\ \text{and } \sigma \in [\![\, t \in Sub(v\delta, E, \mathcal{E}, \mathcal{K}) \,]\!]^e_{\tau'}
\end{array} \right. \qquad \}
\end{array}
$$

$[\![\, t \in Sub_d(w, E, \mathcal{E}, \mathcal{K}) \,]\!]^e_\tau$ *is defined in a similar way as* $[\![\, t \in Sub(w, E, \mathcal{E}, \mathcal{K}) \,]\!]^e_\tau$ *with the difference that for the first case (when* $u\sigma = \overline{t}^e\tau\sigma$*), we have* $u <^L_F \overline{w}^e\tau$*.*

Definition 15 (Solutions of a Negative Constraint).

$$
\begin{array}{ll}
[\![\, X_m \notin Forge_c(E, \mathcal{K}) \,]\!]^e_\tau & = \mathcal{GS} \setminus [\![\, X_m \in Forge_c(E, \mathcal{K}) \,]\!]^e_\tau \\
[\![\, (\forall i\ X_m \neq u) \,]\!]^e_\tau & = \mathcal{GS} \setminus \bigcup_{x=1\ldots e} [\![\, X_m = u \,]\!]^e_{[i \leftarrow x], \tau}
\end{array}
$$

We describe our constraint system by blocks in the following way:

Definition 16 (Constraint System). *First, we define a constraints block* B *as a conjunction of constraints together with an environment* $\mathcal{E}$*:* $B = (ctr_1 \wedge \ldots \wedge ctr_l, \mathcal{E})$*. We will sometimes handle blocks as set of elementary or negative constraints for ease of notations. For instance we write* $c \in B$ *to express that the elementary constraint* c *is a conjunct of* B*.*

We can now define the constraint system that we will use to represent protocol runs. Given two finite lists of index variables $Q = i_1, \ldots, i_k$ *and* $R = j_1, \ldots, j_l$*, we write the quantifier prefix* $\forall i_1 \cdots \forall i_k \exists j_1 \cdots \exists j_l$ *in short:* $\forall Q\ \exists R$*. A constraint system, denoted by* S*, is a disjunction of blocks with a quantifier prefix:* $S = \forall Q\ \exists R\ (B_1 \vee \ldots \vee B_p)$

Now, we define the set of solutions of the constraint system as follows:

Definition 17 (Solutions of the Constraint System). *Consider a constraint system* S*,* B_i*, for* $i = 1 \ldots p$ *(blocks of* S*) and* $ctr_{i,j}$*, for* $j = 1 \ldots l_i$ *(constraints of the block* B_i*) given in Definition 16. The set of solutions of a constraint system* CS *is defined inductively using the following cases:*

$$
\begin{array}{ll}
[\![\, \forall i\ S \,]\!]^e_\tau = \cap_{x=1,\ldots,e} [\![\, S \,]\!]^e_{[i \leftarrow x], \tau} & [\![\, S \,]\!]^e_\tau = \bigcup_{i=1\ldots p} [\![\, B_i \,]\!]^e_\tau \\
[\![\, \exists i\ S \,]\!]^e_\tau = \cup_{x=1,\ldots,e} [\![\, S \,]\!]^e_{[i \leftarrow x], \tau} & [\![\, B_i \,]\!]^e_\tau = \bigcap_{j=1\ldots l_i} [\![\, ctr_{i,j} \,]\!]^e_\tau
\end{array}
$$

The idea will be to use a constraint system based on blocks to represent all possible ways the intruder can construct a list of terms represented by an *mpair*. Roughly, there will be one block in the system for each way. Note also that blocks are extended to admit labeled constraints:

Notation 1 *(labels of constraints)*
A constraint ctr may be equiped with a label $(ctr)^m$ *or* $(ctr)^{sm}$ *or* $(ctr)^f$ *to denote respectively a master constraint or a submaster constraint or a final constraint. The two first labels allow us to keep track of the "official" formal value of some indexed or non-indexed variable. For example, we will prove that we have exactly one master constraint for every indexed variable in each block, and we will use master or sub master constraints to instanciate variables when needed; The third label will be used to prevent any further rewriting on some constraint. We introduce also the notation* $(ctr)^\bullet$ *to refer to labeled or non labeled constraint. The solutions of labeled constraints are the solutions of the constraints obtained by removing labels.*

To simplify the use of (sub)master constraints, we group them into sets:

Definition 18 (Set of (sub)master constraints). *Let* $S = \forall Q \exists R\ B_1 \vee .. \vee B_p$ *be a constraint system,* $Y \in \mathcal{X}$, $W \subseteq \mathcal{X}$ *and* $\overrightarrow{X} \in \overrightarrow{\mathcal{X}}$. *We define* $\mathcal{M}(S, \overrightarrow{X}) = \{ctr \,|\, \exists i\ (ctr)^m \in B_i$ *and* $\exists j \in Q$ *such that* $ctr = (X_j \in Forge_c(E, \mathcal{K}))$ *or* $ctr = (X_j = u)\}$ *and* $\forall i\ \mathcal{SM}(B_i, W) = \{ctr \,|\, (ctr)^{sm} \in B_i,\ ctr = (Y = u)$ *and* $Y \in W\}$. *Also,* $\mathcal{SM}(B_i, Y) = \mathcal{SM}(B_i, \{Y\})$.

When S is clear from the context we omit it in $\mathcal{M}(S, \overrightarrow{X})$ and write simply $\mathcal{M}(\overrightarrow{X})$.

4 Normalisation of a Constraint System

In this section we present the rules applied in the normalization function over constraint systems. The result of applying a rule is put in disjunctive normal form and existential quantifiers are moved up to the prefix of the system using first order logic. These rules are organized in six groups $G_1, \ldots, G_6$.

G_1 aims at maintaining syntactic properties over a constraint system. Some rules handle labelling of master constraints by adding new labels or transfering existing ones. We call them respectively the labelling and the label transfer rules. Other rules format constraints in order to get preferably variables on their left hand-side, or replace indexed variables by non-indexed ones.

G_2 contains the $Forge$ rules. Here, we have for example :

$$t \in Forge(E, \mathcal{K}) \longrightarrow t \in Forge_c(E, \mathcal{K}) \vee \bigvee_{w \in E} t \in Sub(w, E, \mathcal{E}, \mathcal{K})$$

This generic rule illustrates the two possible ways for forging a term t: either by composing or by decomposing one of the knowledge. The other rules enumerate all possible ways a term can be composed by the Intruder: we have exactly one rule for decomposing each kind of operator in the signature $\mathcal{G}$. In particular, there is one rule for the *mpair* operator where *mpair* autonomy is used to justify the quantification.

G_3 contains the Sub rules. These rules are similar to the $Forge$ ones, but they decompose Intruder knowledge. In this group, there is a generic rule :

$$t \in Sub(u, E, \mathcal{E}, \mathcal{K}) \longrightarrow (t = u) \text{ if } u \in \mathcal{K}$$
$$(t = u) \vee (t \in Sub_d(u, E, \mathcal{E}, \mathcal{K})) \text{ otherwise}$$

This rule follows precisely the intruder deduction rules: a term t is an accessible subterm of u iff $t = u$ or there exists a direct subterm u' of u, derivable from u, with t being an accessible subterm of u'. Therefore, there exists exactly one rule for decomposing each kind of operator in $\mathcal{G}$ (apart from variables and constants).

G_4 encodes unification algorithm for terms in our system, and thus, the resolution of equality constraints. These rules simply consist in testing recursively the compatibility of each top operator in each term. Therefore, the only equality constraints remaining after an iteration of these rules are those assigning a value to a variable, i.e. $X = u$ with $X \in \mathcal{X} \cup \mathcal{X}_{\mathcal{I}}$. The goal of G_5 and G_6 is to replace variables by managing interactions between constraints.

G_5 aims at replacing variables by their value, inside one block and independently of other blocks. Consequently, these rules only consider multiple occurences of the same variable, with the same index in case of indexed variable. For instance, the interaction between two $Equality$ constraints is managed by Rules 26 and 27 for respectively indexed and non indexed variables.

$$B \wedge (X_i = u)^{\bullet} \wedge (X_i = v)^{\bullet} \longrightarrow B \wedge (X_i = u)^{\bullet} \wedge (X_i = v)^{\bullet} \wedge u = v \quad (26)$$
$$(X = u)^{sm} \wedge (X = v) \longrightarrow (X = u)^{sm} \wedge (u = v) \quad (27)$$

Besides, interaction for $Forge$ and Sub constraints is managed as follows:

$$A \in Forge_c(E, \mathcal{K}) \wedge t \in Sub_d(A, E', \mathcal{E}, \mathcal{K}) \longrightarrow \bot \text{ where } A \in \mathcal{X} \cup \mathcal{X}_{\mathcal{I}}$$

This rule says that it is not necessary to decompose a variable. While a bit more complex than expected, our semantics of $Sub_d(,,)$ has been defined to prevent useless actions like this, thus ensuring the validity of G_6.

G_6 generalizes G_5 by allowing variable replacements from one block to an other one. Given a constraint containing a variable X_i that must be replaced by its value, we enumerate a finite number of "candidate" terms representing all possible values of this variable according to the whole constraint system. These values are provided by master constraints. For instance, the interleaving between a Sub constraint with contraints of other blocks leads to the following rule:

$$t \in Sub_d(X_m, E', \mathcal{E}, \mathcal{K}) \longrightarrow \bigvee_{(X_i = u) \in \mathcal{E}} \exists k'\ t \in Sub(u\delta, E', \mathcal{E}, \mathcal{K}) \wedge (X_m = u\delta)^f$$
$$\wedge X_m \notin Forge_c(E', \mathcal{K}) \text{with}\, \delta = \delta_{i,m}^{k'}$$

In this rule, only the case where master constraints are *equality* ones are taken into account since the interleaving with $Forge$ ones leads to $\bot$. This rule adds an (extra) equality representing the master constraint it used, but labeled final to prevent further reductions on it. It adds also negative constraints to eliminate the case of $Forge$ master constraints.

Rules System with Tags We have defined our Rules system for constraints on untagged terms. Nevertheless, these rules deal also with tagged terms following the definition of our signature. For example, the equality constraint $(X_i)^i = u$ leads to $\bot$ when u is untagged, since $(X_i)^i = [e_i, X_i]$.

Definition 19 (Solved Constraint). *A solved constraint is of type:* $(X_i = u)^\bullet$, $(X = u)^{sm}$, $(X_i \in Forge_c(E, \mathcal{K}))^\bullet$, $X \in Forge_c(E, \mathcal{K})$, $(Y \notin Forge_c(E, \mathcal{K}))$, *or* $(\forall j\ X_i \neq u)$ *where* $X \in \mathcal{X}$, $Y \in \mathcal{X} \cup \mathcal{X}_\mathcal{I}$, $X_i \in \mathcal{X}_\mathcal{I}$, $u \in \mathcal{T}$, $j \in Var_\mathcal{I}(u)$, $E \subset \mathcal{T}$ *and* $\mathcal{K} \subset \mathcal{T}$.

We will prove that at each step of our algorithm, the normalized constraint system contains only solved constraints.

Application to the Asokan-Ginzboorg Protocol

Let us Focus on the following step of the Asokan-Ginzboorg specification:
$mpair(i, \langle L, \{E_i\}_p\rangle) \in Forge(E_1, \emptyset)$ where $E_1 = \{mpair(t, \langle l, \{e\}_p\rangle)\}$.

$$\begin{aligned}
&\longrightarrow (mpair(i, \langle L, \{E_i\}_p\rangle) \in Forge_c(E_1, \emptyset)) \\
&\quad \vee (mpair(i, \langle L, \{E_i\}_p\rangle) \in Sub(mpair(t, \langle l, \{e\}_p\rangle), E_1, \emptyset, \emptyset)) \text{ by } G_2 \\
&\longrightarrow^* \ \forall i\, \forall j\, ((L \in Forge(E_1, \emptyset) \wedge (E_i = e)^m) \vee ((L = l)^{sm} \wedge (E_i = e)^m) \\
&\quad \vee ((L = l)^{sm} \wedge (E_j = e)^m)) \text{ by } G_1
\end{aligned}$$

5 Well-tagged Protocol Verification

We introduce here the verification algorithm and the results that state the correctness and the completeness of the inference rules of Section 4 and decidability for protocols without $mpair(,)$'s and indexed variables. Given a set R of inference rules and a formula F we say that $R(F)$ is a *closure* of F by R if it is derived by a finite number of applications of rules in R and no rule can be further applied to $R(F)$. First of all, we define a reduction of equalities chain in an environment:

Notation 2 *($\lceil\mathcal{E}\rceil$) We note $\lceil\mathcal{E}\rceil$ the closure of $\mathcal{E}$ by the following rules:*

$$X = Y \wedge Y = u \longrightarrow X = u \wedge Y = u$$
$$X_i = Y_i \wedge Y_j = u \longrightarrow X_i = u\delta^{k'}_{Q,i} \wedge Y_j = u$$

for $X, Y \in \mathcal{X} \cup \mathcal{X}_\mathcal{I}$, $X_i, Y_j \in \mathcal{X}_\mathcal{I}$ *and* $u \notin \mathcal{X} \cup \mathcal{X}_\mathcal{I}$.

Second, we introduce the normalisation function denoted by $S \mapsto (S)\downarrow$. A normalisation of a constraint system can be defined when some closures can be computed as follows using a subset of the inference rules. This normalization operates in two main phases:

Definition 20 (Normalization function). *Let S be a block system. We denote by SR the whole set of inference rules except the labelling rule. We assume that we can compute:*
Phase 1: S_1, a closure of S through SR except Rules 26 *and* 27*;*
Labelling: S_2, a closure of S_1 through the labelling and label transfer rules;
Phase 2: S_3, a closure of S_2 through SR. This closure is denoted by $(S)\downarrow$.

The *Labelling* step adds labels for creating master constraints, making sure to always favour labelling of equality constraints to a forge constraints. While Phases 1 and 2 are similar by the rules they use, their behaviors differ: when used in our algorithm for a step $R_i \Rightarrow S_i$, Phase 1 will never use any constraint interleaving rule with a master constraint $\mathcal{M}(\overrightarrow{X})$ with $\mathcal{L}(X) = E_{i-1}$. This means that during Phase 1, the variables with maximum level cannot be replaced yet from a block to an other, simply because none of them have master constraints yet. However, the second phase do not have this limitation. The verification algorithm is the following:

Algorithm 1 *Let $P = \{R_i \Rightarrow S_i | i = 1..k\}$ be Well-Tagged, $Sec \in \mathcal{T}$, $R_{k+1} \triangleq Sec$ and $S_0 \subset \mathcal{T}_g$.*

1. *Let $CBS_0 \triangleq \forall Q\, \exists R\, \top$, with $Q = R = \emptyset$, be the initial constraint system.*
2. *For i from 1 to $k+1$:*
   (a) *Assume that $CBS_{i-1} \triangleq \forall Q \exists R\, B_1 \vee B_2 \vee .. \vee B_p$;*
   (b) *Let $ctr_i \triangleq R_i \in Forge(S_0, S_1, .., S_{i-1}, \emptyset)$;*
   (c) *Let $\mathcal{E}_i = \bigcup_{\overrightarrow{X}} \mathcal{M}(CBS_{i-1}, \overrightarrow{X})$ and for all $j = 1, ..p$, $X, Y \in \mathcal{X} \cup \mathcal{X}_{\mathcal{I}}$, $\mathcal{E}_{i,j} = \lceil \mathcal{E}_i \cup \mathcal{SM}(B_j, \mathcal{X}) \rceil \backslash \{(X = Y)\}$;*
   (d) *Let $CBS_i \triangleq (\forall Q \exists R\ (B_1 \wedge ctr_i, \mathcal{E}_{i,1}) \vee .. \vee (B_p \wedge ctr_i, \mathcal{E}_{i,p}))\downarrow$*
3. *Test Satisfiability of CBS_{k+1} (return insecure iff satisfiable).*

Note that sets $\mathcal{E}_i, \mathcal{E}_{i,j}$ denotes respectively the set of master constraints for vector variables and the set of submaster constraints for variables of block B_j, both with variables of level strictly included in E_{i-1}. Notation $\top$ represents true. The algorithm chooses a "possible" protocol run represented by π, and tests if after this run Sec is derivable by the intruder for some length e of $mpair(,)$. We test this by increasing the initial constraint system CBS_0 with each protocol step successively, and by normalising the resulting constraint system at each step. This step-by-step normalisation is required by our inference rules which assumes that master and sub-master constraints for previous steps have been already computed. Since we will show that the normalisation preserves the semantics, $[\![CBS_{k+1}]\!]^e_\emptyset \neq \emptyset$ iff the protocol run defined by π has an attack. Thus, assuming the next lemmas, the correctness of the algorithm follows:

Lemma 1. *(Correctness and Completeness of Normalization)*
Let CBS_i and ctr_i ($i = 1..k+1$) be as in the verification algorithm, for some Well-tagged *protocol P. Then for all e,*

$$[\![CBS_{i-1} \wedge ctr_i]\!]^e_\emptyset = \big|\!\big[(CBS_{i-1} \wedge ctr_i)\downarrow \big]\!\big|^e_\emptyset$$

Lemma 2. *(Satisfiability of normalised form)*
If Algorithm 1 terminates for a subclass of Well-Tagged protocols, then the satisfiability of a normalised constraint system resulting from a protocol in this subclass is decidable.

Theorem 2. *(Analysis of Well-tagged Protocols)*
If Algorithm 1 terminates for a subclass of Well-Tagged protocols, then, the insecurity problem is decidable for this subclass.

Full proofs can be found in Technical report [Rep]. Moreover, it is worth to notice that our algorithm always terminates for protocols without $mpair(,)$, thus showing that our procedure is an extension of protocol analysis in the basic case. The proof of termination for protocols without $mpair(,)$ and without indexed variables can be found in Technical report [Rep].

6 Conclusion and Further Works

We have proposed an extension of the constraint-based approach in symbolic protocol verification in order to handle a class of protocols (the well-tagged ones) which admit unbounded lists in messages. This class can be used to model in particular interesting group protocols. We conjecture that adding adequate control on our constraint simplication rules allows one to obtain termination of the constraint normalization process and therefore to derive a decision procedure for an interesting subclass of well-tagged protocols.

References

[AC02] A. Armando and L. Compagna. Automatic sat-compilation of protocol insecurity problems via reduction to planning. In *Foundation of Computer Security & Verification Workshops*, Copenhagen, Denmark, July 25-26 2002.

[AG00] N. Asokan and P. Ginzboorg. Key agreement in ad hoc networks. *Computer Communications*, 23(17):1627–1637, 2000.

[BMV03] D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In Einar Snekkenes and Dieter Gollmann, editors, *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, 2003.

[BO97] J.A. Bull and D.J. Otway. The authentication protocol. Technical report, Defence Research Agency, Mavern,UK, 1997.

[BP03] B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In Andrew D. Gordon, editor, *FOSSACS*, volume 2620 of *Lecture Notes in Computer Science*, pages 136–152, Warsaw, Poland, April 7-11 2003. Springer.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. Inform Theory IT-29*, pages 198–208, 1983. Also STAN-CS-81-854, May 1981, Stanford U.

[KKW07] K.O. Kürtz, R. Küsters, and T. Wilke. Selecting theories and nonce generation for recursive protocols. In *FMSE '07: Proceedings of the 2007 ACM workshop on Formal methods in security engineering*, pages 61–70, New York, NY, USA, 2007. ACM.

[KMT08] S. Kremer, A. Mercier, and R. Treinen. Proving group protocols secure against eavesdroppers. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR'08)*, Lecture Notes in Artificial Intelligence, Sydney, Australia, August 2008. Springer-Verlag. To appear.

[KT07] R. Küsters and T. Truderung. On the automatic analysis of recursive security protocols with xor. Technical report, ETH Zurich, 2007. An abridged version appears in STACS 2007.

[KW04] R. Küsters and T. Wilke. Automata-based analysis of recursive cryptographic protocols. In *21st Symposium on Theoretical Aspects of Computer Science (STACS 2004)*, Lecture Notes in Computer Science. Springer-Verlag, 2004.

[Mea00] C. Meadows. Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In P.Degano, editor, *the First Workshop on Issues in the Theory of Security*, pages 87–92, Geneva, Switzerland, July 2000.

[Mit97] S. Mittra. Iolus: A framework for scalable secure multicasting. In *SIGCOMM*, pages 277–288, 1997.

[MS01] C. Meadows and P. Syverson. Formalizing gdoi group key management requirements in npatrl. In *CCS'01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 235–244, New York, USA, 2001. ACM Press.

[MT07] J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. *Higher-Order and Symbolic Computation*, 20(1-2):123–160, 2007.

[Pau96] L. C. Paulson. Isabelle: A generic theorem prover. *Lecture Notes in Computer Science*, 828:283–298, 1996.

[Pau97] L. C. Paulson. Mechanized proofs for a recursive authentication protocol. In *10th Computer Security Foundations Workshop*, pages 84–95. IEEE Computer Society Press, 1997.

[PQ03] O. Pereira and J.-J. Quisquater. Some attacks upon authenticated group key agreement protocols. *Journal of Computer Security*, 11(4):555–580, 2003.

[PQ04] O. Pereira and J.-J. Quisquater. Generic insecurity of cliques-type authenticated group key agreement protocols. In *CSFW*, pages 16–19, 2004.

[Rep] http://www.loria.fr/~turuani/TR-PCP08.pdf.

[RS03] R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable with unbounded nonces as well. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FST TCS, Proceedings*, volume 2914 of *Lecture Notes in Computer Science*, pages 363–374, Mumbai, India, December 15-17 2003. Springer.

[RT03] M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions, composed keys is np-complete. *Theor. Comput. Sci.*, 1-3(299):451–475, 2003.

[SB04] G. Steel and A. Bundy. Attacking group multicast key management protocols using CORAL. In A. Armando and L. Viganó, editors, *Proceedings of the ARSPA Workshop*, volume 125 of *ENTCS*, pages 125–144, 2004.

[SWT98] M. Steiner, M. Waidner, and G. Tsudik. Cliques: A new approach to group key agreement. In *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, page 380, Washington, DC, USA, 1998. IEEE Computer Society.

[TJ03] M. Taghdiri and D. Jackson. A lightweight formal analysis of a multicast key management scheme. In *FORTE*, pages 240–256, 2003.

[Tru05] T. Truderung. Selecting theories and recursive protocols. *Lecture Notes in Computer Science*, pages 217–232, 2005.

[Wei99] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *16th International Conference on Automated Deduction*, volume 1632 of *lncs*, pages 314–328. Springer, 1999.